Exhibit 21 to Amended Complaint
Intellectual Ventures I LLC and Intellectual Ventures II LLC

Example American Count VII Systems and Services
U.S. Patent No. 7,712,080 ("the '080 Patent")

The Accused Systems and Services include without limitation American systems and services that utilize Spark; all past, current, and future systems and services that operate in the same or substantially similar manner as the specifically identified systems and services; and all past, current, and future American systems and services that have the same or substantially similar features as the specifically identified systems and services ("Example American Count VII Systems and Services" or "American Systems and Services").[1]

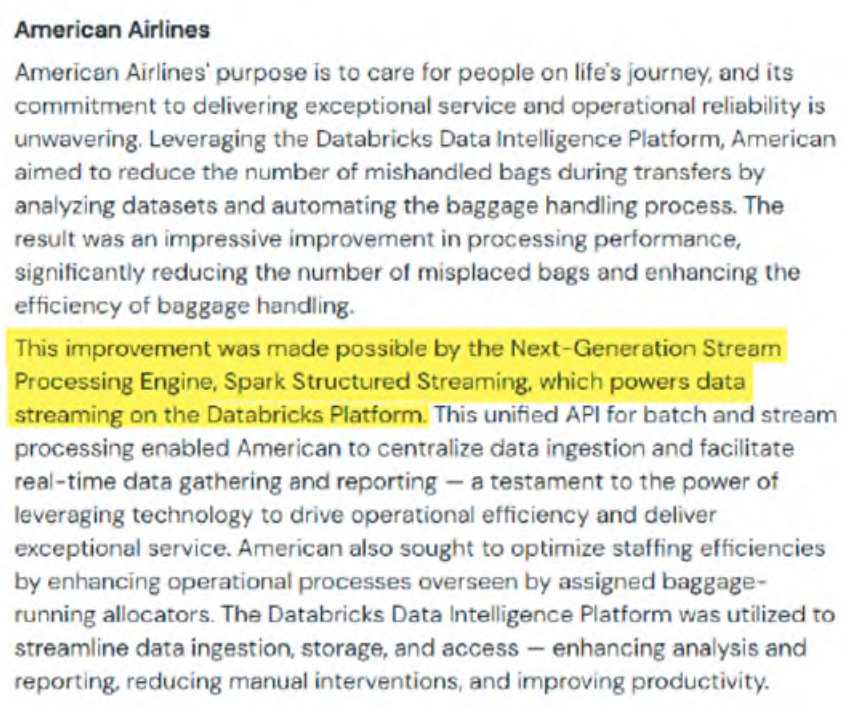
On information and belief, the American Systems and Services use Spark in public and/or private cloud(s). For example, American posts, or has posted, job opportunities that require familiarity with Spark technology.

- Example of a Senior Java Full Stack Developer at American Airlines whose position mentions data transformation and streaming development via Spark. https://www.linkedin.com/in/rohitha-m6363/ (Last accessed on 9/19/24)
- Example of a Big Data Engineer at American Airlines whose position mentions the use of Spark. https://www.linkedin.com/in/krishna-karthika-katragadda-149786306/ (Last accessed on 9/19/24)
- Example of a Senior Data Engineer at American Airlines whose position mentions analyzing and developing programs, writing programs, and developing jobs via Spark. https://www.linkedin.com/in/sriram1993/ (Last accessed on 9/19/24)
- Example of a Senior Data Engineer at American Airlines whose position mentions the use of Spark. https://www.linkedin.com/in/madhu-sudhan-reddy-y-b3897b129/ (Last accessed on 9/19/24)
- Example of a Data Engineer at American Airlines whose position mentions the use of Spark. https://www.linkedin.com/in/ashish-n-013372287/ (Last accessed on 9/19/24)
- Example of Analyst/Sr Analyst, Safety Data Analytics Job Listing at American Airlines which mentions use of Spark. https://jobs.aa.com/job/AnalystSr-Analyst%2C-Safety-Data-Analytics/74823-en_US (Last accessed on 9/19/24)

[1] For the avoidance of doubt, Plaintiffs do not accuse the public clouds of Defendant, to the extent those services are provided by a cloud provider with a license to Plaintiffs' patents that covers Defendant's activities. As a specific example, Plaintiffs do not accuse Amazon managed services, *i.e.*, Amazon Elastic Kubernetes Service (Amazon EKS) and Amazon Elastic Container Service (Amazon ECS). Plaintiffs also does not accuse IBM managed services, *i.e.*, Red Hat Open Shift. Plaintiffs do not accuse the public clouds of Defendants if those services are provided by a cloud provider with a license to Plaintiffs' patents that covers Defendants' activities. Plaintiffs will produce relevant license agreements in this litigation. Plaintiffs accuse Defendant private clouds that implement Spark and non-licensed public clouds that Defendant uses to support Spark for its systems and services. Plaintiffs will provide relevant license agreements for cloud providers in discovery, to the extent any such license agreements have not already been produced. To the extent any of these licenses are relevant to Defendant's activities, Plaintiffs will meet and confer with Defendant about the impact of such license(s).

As another example, American has announced cloud migration of legacy technology and efforts to modernize its mainframes and servers. Source: https://dxc.com/sg/en/insights/customer-stories/american-airlines-cloud-data-automation. American continues to use private cloud for at least certain applications. Source: https://www.techtarget.com/searchdatamanagement/feature/American-Airlines-lowers-data-management-costs-with-Intel ("American Airlines' initial target for cost optimization was Azure Data Lake, according to Vijay Premkumar, senior manager of public and *private cloud* at the airline.") (emphasis added).[2]

On information and belief, other information confirms American uses Spark technology.



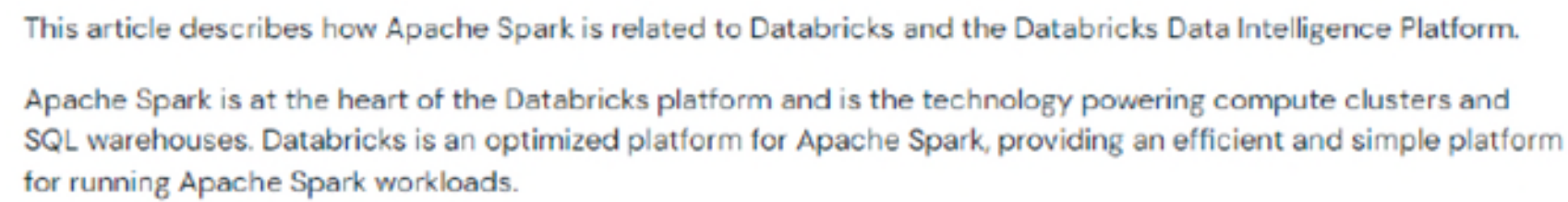
American Airlines

American Airlines' purpose is to care for people on life's journey, and its commitment to delivering exceptional service and operational reliability is unwavering. Leveraging the Databricks Data Intelligence Platform, American aimed to reduce the number of mishandled bags during transfers by analyzing datasets and automating the baggage handling process. The result was an impressive improvement in processing performance, significantly reducing the number of misplaced bags and enhancing the efficiency of baggage handling.

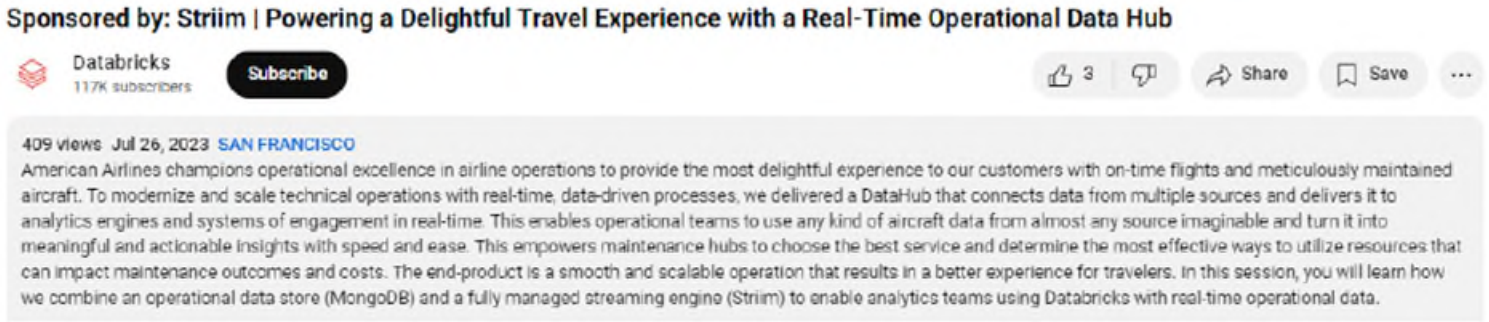
This improvement was made possible by the Next-Generation Stream Processing Engine, Spark Structured Streaming, which powers data streaming on the Databricks Platform. This unified API for batch and stream processing enabled American to centralize data ingestion and facilitate real-time data gathering and reporting — a testament to the power of leveraging technology to drive operational efficiency and deliver exceptional service. American also sought to optimize staffing efficiencies by enhancing operational processes overseen by assigned baggage-running allocators. The Databricks Data Intelligence Platform was utilized to streamline data ingestion, storage, and access — enhancing analysis and reporting, reducing manual interventions, and improving productivity.

Source: https://www.databricks.com/blog/disrupting-status-quo-through-data-and-ai-celebrating-2024-data-team-disruptor-award-nominees.

[2] Unless otherwise noted, all sources cited in this document were publicly accessible as of the date of the Amended Complaint.



Source: https://docs.databricks.com/en/spark/index.html.



Source: https://www.youtube.com/watch?v=NmdBiO_U7rA.

Source: https://www.databricks.com/training/catalog/databricks-performance-optimization-1809.

U.S. Patent No. 7,712,080 (Claim 9)	
Claim(s)	**Example American Count VII Systems and Services**
[9.pre]. A distributed parallel computing system, having at least one memory area and at least one processor, comprising:	To the extent this preamble is limiting, on information and belief, the American Count VII Systems and Services include a distributed parallel computing system having at least one memory area and at least one processor. For example, Spark consists of a distributed parallel computing system having multiple memory and processors. **Overview** At a high level, every Spark application consists of a *driver program* that runs the user's `main` function and executes various *parallel operations* on a cluster. The main abstraction Spark provides is a *resilient distributed dataset* (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel. RDDs are created by starting with a file in the Hadoop file system (or any other Hadoop-supported file system), or an existing Scala collection in the driver program, and transforming it. Users may also ask Spark to *persist* an RDD in memory, allowing it to be reused efficiently across parallel operations. Finally, RDDs automatically recover from node failures. Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html. **Which Storage Level to Choose?** Spark's storage levels are meant to provide different trade-offs between memory usage and CPU efficiency. We recommend going through the following process to select one: • If your RDDs fit comfortably with the default storage level (`MEMORY_ONLY`), leave them that way. This is the most CPU-efficient option, allowing operations on the RDDs to run as fast as possible. • If not, try using `MEMORY_ONLY_SER` and selecting a fast serialization library to make the objects much more space-efficient, but still reasonably fast to access. (Java and Scala) • Don't spill to disk unless the functions that computed your datasets are expensive, or they filter a large amount of the data. Otherwise, recomputing a partition may be as fast as reading it from disk. • Use the replicated storage levels if you want fast fault recovery (e.g. if using Spark to serve requests from a web application). *All* the storage levels provide full fault tolerance by recomputing lost data, but the replicated ones let you continue running tasks on the RDD without waiting to recompute a lost partition. Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html.

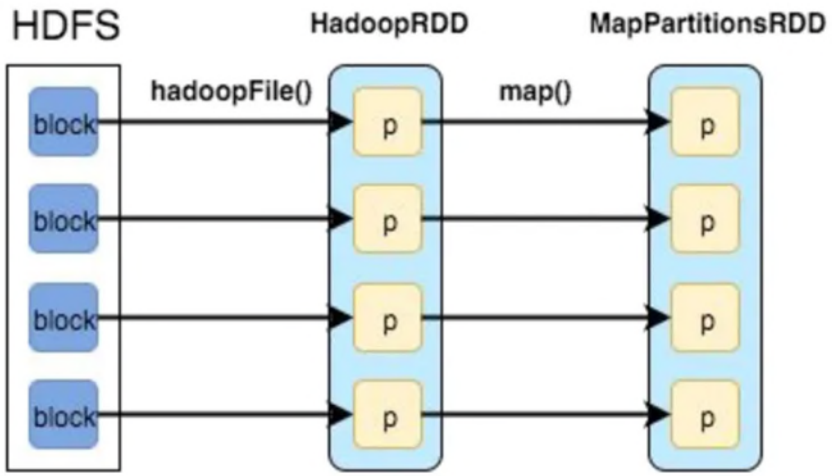
U.S. Patent No. 7,712,080 (Claim 9)	
Claim(s)	**Example American Count VII Systems and Services**
[9.a] at least one distributed shared variable capable of loading into the at least one memory area, wherein the at least one distributed shared variable is a single variable that includes several variables that may be physically loaded into the at least one memory area; and	On information and belief, the American Count VII Systems and Services include at least one distributed shared variable capable of loading into the at least one memory area, wherein the at least one distributed shared variable is a single variable that includes several variables that may be physically loaded into the at least one memory area. For example, Spark collects large volumes of data and distributes the data across multiple nodes. Spark automatically partitions the data into Resilient Distributed Datasets (RDDs) that operate in parallel. The RDDs are partitioned across multiple nodes and processed across at least one memory area. Resilient Distributed Datasets (RDDs) Spark revolves around the concept of a *resilient distributed dataset* (RDD), which is a fault-tolerant collection of elements that can be operated on in parallel. There are two ways to create RDDs: *parallelizing* an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared filesystem, HDFS, HBase, or any data source offering a Hadoop InputFormat. Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html.

U.S. Patent No. 7,712,080 (Claim 9)	
Claim(s)	**Example American Count VII Systems and Services**
	RDDs — Partitions • RDDs are designed to contain huge amounts of data , that cannot fit onto one single machine → hence, the data has to be partitioned across multiple machines/nodes • Spark automatically partitions RDDs and distributes the partitions across different nodes • A partition in spark is an atomic chunk of data stored on a node in the cluster • Partitions are basic units of parallelism in Apache Spark • RDDs in Apache Spark are collection of partitions • Operations on RDDs automatically place tasks into partitions, maintaining the locality of persisted data Source: https://blog.k2datascience.com/batch-processing-apache-spark-a67016008167. **Parallelism** Parallelism in Spark is all about doing multiple things at the same time. The core idea is to divide the data into smaller chunks (partitions), so they can be processed simultaneously. The more partitions you have, the more tasks can run in parallel, leading to faster processing times. Source: https://dataengineerinterview.substack.com/p/a-deep-dive-into-apache-spark-partitioning.

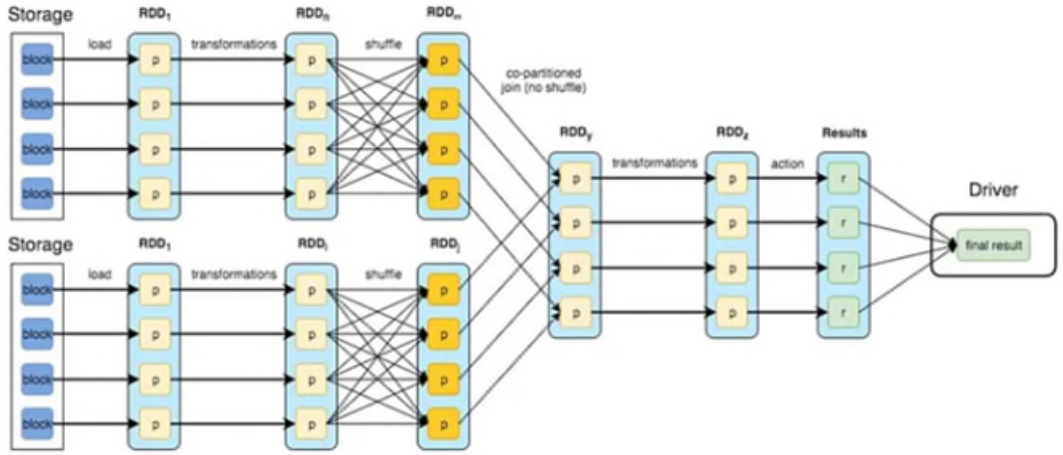
U.S. Patent No. 7,712,080 (Claim 9)	
Claim(s)	**Example American Count VII Systems and Services**
	Imagine you have a dataset with 1 million records. If this dataset is divided into 10 partitions, Spark can launch 10 tasks to process each partition simultaneously. If the cluster has sufficient resources, these tasks will run in parallel, speeding up the process. However, if you increase the number of partitions to 100, and assuming your cluster has the resources (like CPU cores) to handle these tasks, Spark can now launch 100 tasks to run concurrently. This dramatically increases parallelism and reduces the total time taken to process the entire dataset. Source: https://dataengineerinterview.substack.com/p/a-deep-dive-into-apache-spark-partitioning. **In-Memory Processing: Spark's Secret Sauce** The magic ingredient that sets Apache Spark apart from many other data processing frameworks is its effective utilization of in-memory processing. In-memory processing involves storing and processing data directly in a computer's RAM (Random Access Memory), as opposed to reading and writing data to slower storage devices like hard drives. Source: https://medium.com/@harshavardhan7696/unleashing-speed-and-efficiency-in-memory-processing-in-apache-spark-c9e7ccc6460e. The RDDs contain variables that allow the nodes to process data in parallel. RDDs are processed using functions and variables that can be used to create other RDDs to further process the data set.

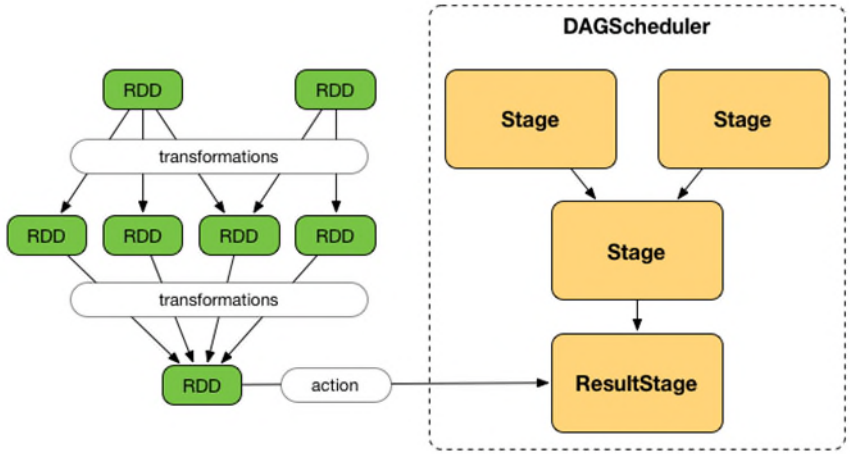
In-Memory Computation

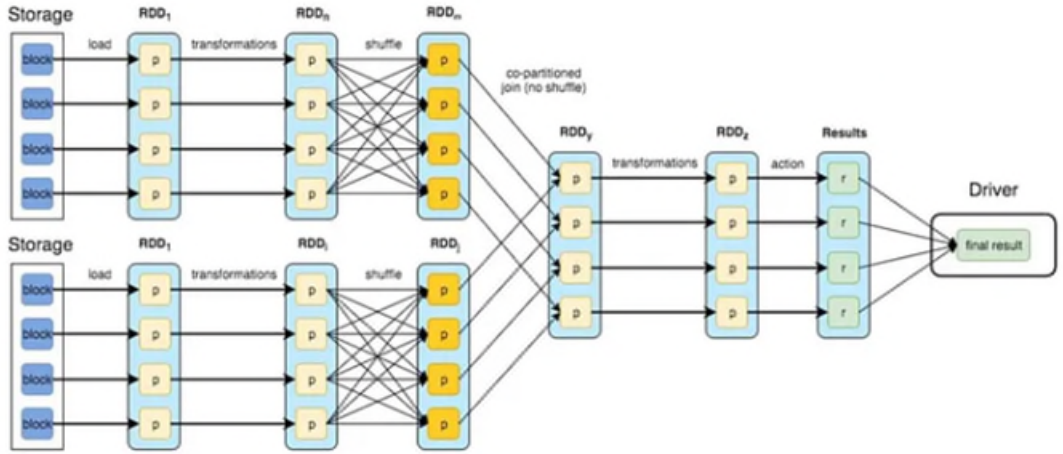
Spark uses in-memory computation as a way to speed up the total processing time. In the in-memory

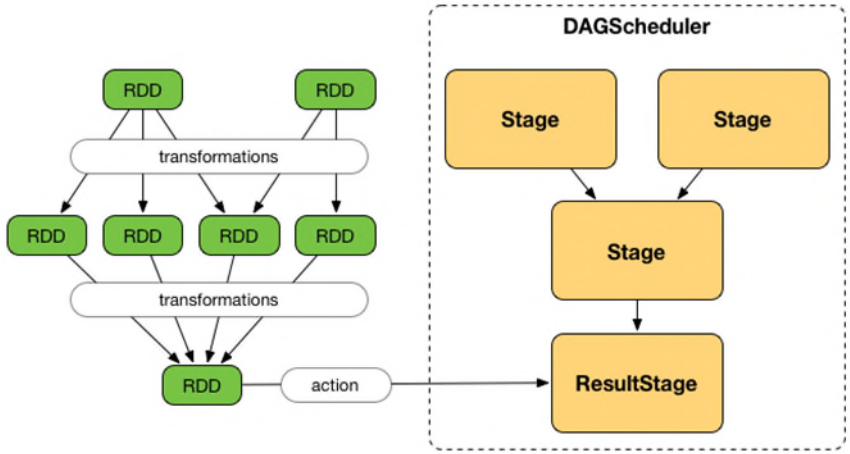
U.S. Patent No. 7,712,080 (Claim 9)	
Claim(s)	**Example American Count VII Systems and Services**
	RDD Creation Here's an example of RDDs created during a method call: Which first loads HDFS blocks in memory and then applies map() function to filter out keys creating two RDDs:  Source: https://blog.k2datascience.com/batch-processing-apache-spark-a67016008167.
[9.b] at least one distributed sequential computing program, configured to operate in the at least one processor, configured to access the at least one distributed shared variable,	On information and belief, the American Count VII Systems and Services include at least one distributed sequential computing program configured to operate in the at least one processor, access the at least one distributed shared variable, and transform into at least one distributed parallel computing program by spawning at least one child distributed sequential computing system program when at least one intermediate condition occurs within the at least one distributed sequential program.

U.S. Patent No. 7,712,080 (Claim 9)	
Claim(s)	**Example American Count VII Systems and Services**
and configured to transform into at least one distributed parallel computing program by spawning at least one child distributed sequential computing system program when at least one intermediate condition occurs within the at least one distributed sequential program,	For example, Spark contains a driver program that is configured to operate in the at least one processor and is configured to access the RDDs. Spark jobs are authored in programming languages such as Java, Scala, and Python, and Spark is configured to access the at least one distributed shared variable (RDDs) through the access of RDDs by a Spark job. **Spark Overview** Apache Spark is a fast and general-purpose cluster computing system. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Spark Streaming. Source: https://archive.apache.org/dist/spark/docs/2.0.1/. **Overview** At a high level, every Spark application consists of a *driver program* that runs the user's `main` function and executes various *parallel operations* on a cluster. The main abstraction Spark provides is a *resilient distributed dataset* (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel. RDDs are created by starting with a file in the Hadoop file system (or any other Hadoop-supported file system), or an existing Scala collection in the driver program, and transforming it. Users may also ask Spark to *persist* an RDD in memory, allowing it to be reused efficiently across parallel operations. Finally, RDDs automatically recover from node failures. Source: https://spark.apache.org/docs/latest/rdd-programming-guide.html. The Spark Execution Model transforms the distributed sequential computing program into a series of stages and sets of parallel tasks that run in parallel on the Spark cluster. This includes Spark tasks operating on a set of RDDs in parallel for a given stage.

U.S. Patent No. 7,712,080 (Claim 9)	
Claim(s)	**Example American Count VII Systems and Services**
	Spark — Execution Workflow  Source: https://blog.k2datascience.com/batch-processing-apache-spark-a67016008167. Spark implements shuffle boundaries where stages and/or tasks wait for the prior stage to complete before fetching map outputs.

U.S. Patent No. 7,712,080 (Claim 9)	
Claim(s)	**Example American Count VII Systems and Services**
	DAGScheduler splits up a job into a collection of stages. Each stage contains a sequence of narrow transformations that can be completed without shuffling the entire data set, separated at **shuffle boundaries**, i.e. where shuffle occurs. Stages are thus a result of breaking the RDD graph at shuffle boundaries.  Figure 4. Graph of Stages Shuffle boundaries introduce a barrier where stages/tasks must wait for the previous stage to finish before they fetch map outputs. Source: https://mallikarjuna_g.gitbooks.io/spark/content/spark-dagscheduler-stages.html.
[9.b.i] wherein the at least one distributed parallel computing program concurrently uses the at least one distributed sequential computing program and the at least one spawned child distributed sequential computing program to	On information and belief, the American Count VII Systems and Services include the at least one distributed parallel computing program concurrently uses the at least one distributed sequential computing program and the at least one spawned child distributed sequential computing program to perform parallel processing and/or operations. For example, Spark implements a set of tasks running in parallel for a given stage. The distributed parallel computing program includes the Spark job and driver program, as described in this claim chart. The spawned child distributed sequential computing program is the sequential code in a task for a given stage, as described in this claim chart. *See* claim limitations [9.b] and [9.b.i-iii].

U.S. Patent No. 7,712,080 (Claim 9)	
Claim(s)	**Example American Count VII Systems and Services**
perform parallel processing and/or operations,	**Spark — Execution Workflow**  Source: https://blog.k2datascience.com/batch-processing-apache-spark-a67016008167.
[9.b.ii] wherein the at least one intermediate condition comprising one intermediate result that will be required by the at least one spawned child distributed sequential computing program to continue computation.	On information and belief, the American Count VII Systems and Services include the at least one intermediate condition comprising one intermediate result that will be required by the at least one spawned child distributed sequential computing program to continue computation. For example, Spark implements shuffle boundary where stages and/or tasks wait for the prior stage to complete before fetching map outputs.

U.S. Patent No. 7,712,080 (Claim 9)	
Claim(s)	**Example American Count VII Systems and Services**
	DAGScheduler splits up a job into a collection of stages. Each stage contains a sequence of narrow transformations that can be completed without shuffling the entire data set, separated at **shuffle boundaries**, i.e. where shuffle occurs. Stages are thus a result of breaking the RDD graph at shuffle boundaries.  Figure 4. Graph of Stages Shuffle boundaries introduce a barrier where stages/tasks must wait for the previous stage to finish before they fetch map outputs. Source: https://mallikarjuna_g.gitbooks.io/spark/content/spark-dagscheduler-stages.html.